Exploiting Node Connection Regularity for DHT Replication

Alessio Pace
Grenoble University & INRIA
alessio.pace@inria.fr

Vivien Quéma
CNRS
vivien.quema@inria.fr

Valerio Schiavoni
University of Neuchâtel
valerio.schiavoni@unine.ch

***Abstract*—Distributed Hash-Tables (DHTs) provide an efficient way to store objects in large-scale peer-to-peer systems. To guarantee that objects are reliably stored, DHTs rely on replication. Several replication strategies have been proposed in the last years. The most efficient ones use predictions about the availability of nodes to reduce the number of object migrations that need to be performed: objects are preferably stored on highly available nodes.**

This paper proposes an alternative replication strategy. Rather than exploiting highly available nodes, we propose to leverage nodes that exhibit regularity in their connection pattern. Roughly speaking, the strategy consists in replicating each object on a set of nodes that is built in such a way that, with high probability, at any time, there are always at least k nodes in the set that are available. We evaluate this replication strategy using traces of two real-world systems: eDonkey and Skype. The evaluation shows that our regularity-based replication strategy induces lower network usage than existing state of the art replication strategies, and better balances the replication and storage loads than existing availability-based approaches.

I. Introduction

Distributed Hash Tables (DHTs) [1] [2] provide a simple high-level put/get abstraction that can be used to build efficient distributed storage systems (e.g. [3] [4]). DHTs have gained wide popularity in the last decade, fostering a large amount of interest in the academia, and inspiring the design of key/value distributed storage systems deployed in production (e.g. [5] [6]).

DHTs provide a way to deterministically map objects to nodes and allow efficiently retrieving objects in a distributed fashion. Nodes and objects are logically arranged in a large numeric key-space, according to a given variant of consistent hashing [7]. Typically, the node in charge of an object is the one whose position immediately follows the object in the key-space.

To ensure the availability of objects despite churn, DHTs rely on replication [8]: multiple copies of each object are stored on different nodes (called *replicas*). A replication protocol is in charge of ensuring that, at any time, each object is replicated on a sufficiently large number of replicas (called the *replica set*). The replication protocol is also in charge of deciding where replicas should be localized. In the past years, several replication strategies have been proposed [8] [9]. The first replication strategies were quite basic: replicas were simply placed on a contiguous set of k online nodes on the DHT [3]. Nevertheless, this simple strategy consumes a very high bandwidth to migrate replicas whenever a node arrival or departure breaks the contiguous set of k replicas. Moreover, as pointed out in [10], bandwidth – and not disk space – is the crucial and limiting factor when designing distributed storage systems. Subsequent works have thus focused on designing replication strategies able to limit the number of object migrations in case of churn. Towards that goal, the most recent works are those leveraging some knowledge about node availability. For instance, in [11], the authors propose node availability predictors which drive the replication strategy to place and migrate objects on the most available nodes. This solution clearly aims at reducing the number of object migrations, at the expense of a fair balance of load on the nodes. Instead, in a recent work [12], the authors propose to build replica sets using nodes with different availability patterns. The goal of this replication strategy is to decrease the number of object migrations, while ensuring a fair balance of the replication load on replicas. Unfortunately, as we show in the evaluation section, this strategy still induces a very high bandwidth consumption.

In this paper, we propose a new approach to replicating objects in DHTs. Rather than focusing on node availability, we propose to leverage nodes that exhibit *regularity* in their connection pattern. To the best of our knowledge, our work is the first attempt at leveraging connection regularity to improve replication in DHTs. Our work is motivated by recent studies that have shown that diurnal availability patterns are commonly exhibited by real-world systems [11] [13] [14] [15]. The key idea is to create for each object a candidate set containing nodes exhibiting regular connection patterns such that, with a high probability, there will always be k online nodes

in the candidate set. The candidate set is then used to feed the replica set. That way, these are always the same set of nodes that will periodically belong to the replica set of a given object. Nodes can thus cache objects and drastically reduce the bandwidth usage. Moreover, this strategy allows leveraging nodes that are not highly available but that are regular. This results in a better replication and storage load balancing than strategies relying on the availability of nodes.

We evaluate our new replication strategy using two traces of real-world systems: eDonkey and Skype. Our evaluation shows that our regularity-based replication strategy induces lower network usage, and better balances the replication and storage loads than existing availability-based approaches.

The remaining of the paper is organized as follows: in Section II, we describe the existing replication strategies. In Section III, we present *Regular*, our new regularity-based replication strategy. We then evaluate the performance of the state-of-the-art strategies of *Regular* in Section IV. We finally discuss related work in Section V, before concluding the paper in Section VI.

II. Background

In this section, we present the state-of-the-art techniques that have been proposed for replicating objects in a DHT. We assume that the DHT is organized as a ring and that each node has a unique identifier that is used to position the node in the DHT (by increasing identifier). Moreover, we assume that each object has a hash value belonging to the same space as the node identifiers. Each replication technique used in this section seeks to maintain k *online* replicas of every object in the DHT. At any point in time, the k nodes that are replicating an object form the *replica set* for that object.

A. Standard

In the *Standard* replication strategy (e.g. [3] [4]) (see Figure 1), the replica set of an object comprises the first k nodes with larger identifiers than the hash of the object. A replica set is updated whenever a replica leaves or a node join causes the first k successors of an object id to change.

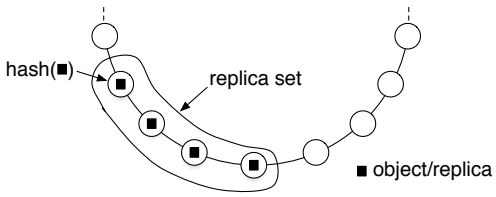


Fig. 1: Standard replication strategy ($k = 4$).

B. Sticky

The *Sticky* replication strategy [16], [11], [17] (see Figure 2) is an extension of the *Standard* strategy whose goal is to avoid migrating objects whenever one of the k successors of an object id changes. The *Sticky* replication strategy works as follows: when an object is created, its replica set first comprises the first k nodes with larger identifiers than the hash of the object (this is similar to the standard strategy). Contrarily to the *Standard* strategy, a replica is used until it leaves the DHT, at which point it is replaced with the first node having a larger identifier than the hash of the object and not belonging to the replica set of the object. A consequence is that the replica set of an object does not necessarily contain the k nodes with larger identifiers than the hash of the object. In order to allow retrieving the replica set of an object, the first k nodes with larger identifiers than the hash of the object store *localization metadata* describing the localization of the current replicas of the object. These metadata are updated whenever the replica set changes.

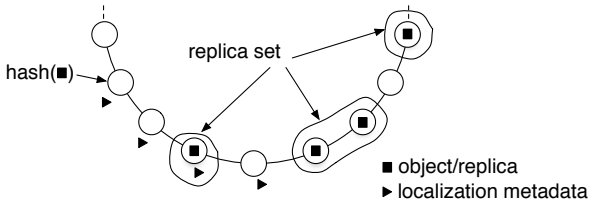


Fig. 2: Sticky replication strategy ($k = 4$).

C. Most-available

The *Most-available* replication strategy [11] (see Figure 3) is an extension of the *Standard* strategy whose goal is to reduce the number of object migrations by choosing highly available nodes to compose replica sets. Replication works as follows: at any time, an object is replicated on the k most available nodes chosen among the set of the S first nodes that have a larger identifier than the hash of the object. This strategy requires to be able to predict the availability of nodes. Efficient heuristics to predict node availability have been proposed in [11]. Note that to retrieve the replicas of an object, it is either possible to query the S nodes having a larger identifier than the hash of the object, or to use *localization metadata* as described in the *Sticky* replication strategy.

D. Sticky most-available

The *Sticky most-available* replication strategy [11] (see Figure 4) is an extension of the *Most-available*

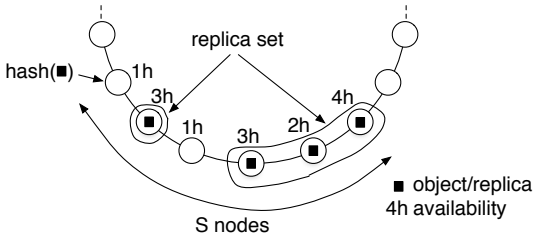


Fig. 3: Most-available replication strategy ($k = 4$).

strategy that aims at reducing the number of required object migrations by relaxing the constraint on the localization of replicas. Replication works as follows: when an object is created, its replica set first comprises the k most available nodes chosen among the set of the S first nodes that have a larger identifier than the hash of the object. Similarly to the *Most-available* strategy, this strategy requires being able to predict the availability of nodes. Contrarily to the *Most-available* strategy, a replica does not need to remain during all its lifetime in the set of the S first nodes with a larger identifier than the hash of the replicated object. Consequently, a replica can be used until it leaves the DHT, at which point it is replaced by the most available node that is both among the S first nodes that have a larger identifier than the hash of the object, and not already in the replica set of the object. Similarly to the *Sticky* strategy, to retrieve the replicas of an object, the *Sticky most-available* strategy makes use of *localization metadata* describing the localization of the current replicas of the object. These metadata are replicated on the first k nodes with larger identifiers than the hash of the object.

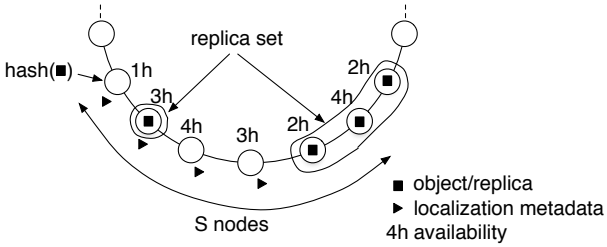


Fig. 4: Sticky most-available replication strategy ($k = 4$).

E. Anti-correlated availability

The *Anti-correlated availability* replication strategy [12] (see Figure 5) aims at reducing the number of object migrations, while balancing the replication load on nodes. For that purpose, the replica set of an object comprises a set of pairs of nodes containing a node p selected at random, and a node q whose availability is anti-correlated with that of node p. Roughly speaking, two nodes have anti-correlated availabilities if one is online while the other one is offline, and vice versa. Choosing a node at random in each pair allows balancing the replication load on nodes, while associating two anti-correlated nodes in each pair allows reducing the number of object migrations (pairs of anti-correlated nodes have a larger "online time" coverage than pairs of randomly selected nodes). Similarly to the *Sticky* replication strategy, it is necessary to use *localization metadata* describing the localization of object replicas. These metadata are replicated on the first k nodes with larger identifiers than the hash of the object.

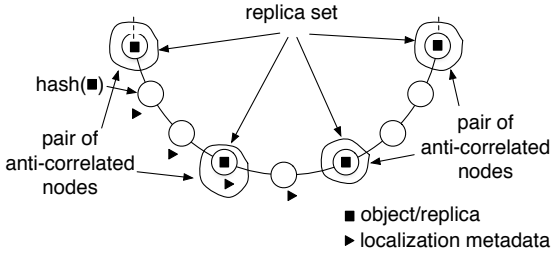


Fig. 5: Anti-correlated availability replication strategy ($k = 4$).

III. Regularity-based replication

In this section, we present a new replication strategy, called *Regular*. We start by motivating the need for a new replication strategy. We then show that nodes in real-world systems exhibit regular connection patterns, a necessary property for the *Regular* replication strategy. Finally, we discuss how we implemented *Regular*.

A. The case for regularity-based replication

All the replication strategies discussed in Section II *reactively* adjust the replica set of an object: when a current replica leaves the network, a new one is picked to ensure that k replicas are currently online. This induces network traffic: online nodes communicate to find a suitable replica and they migrate the replicated object on the new replica.

To reduce the aforementioned costs, existing DHTs let nodes cache the objects they have recently replicated. This solution efficiently limits the number of object migrations when few nodes intermittently join and leave the network. Nevertheless, this solution is inefficient when there is a high level of churn. Indeed, under such conditions, nodes have a low probability to belong multiple times to the same replica set.

In this paper, we propose a way to leverage object caching. More precisely, we propose to *proactively* associate with each object a static *candidate set* of nodes. This *candidate set* should be such that at any given time, there be a high probability that k nodes belonging to the set be online. When such k nodes exist, they are used to form the replica set of the object. Otherwise, the replica set is fed with "temporary" nodes (details are provided in Section III-C). This solution allows reducing the number of object migrations that need to be performed by limiting the set of nodes which can become replicas for a given object (and by thus forcing the same nodes to be regularly in the replica set of the object).

Forming candidate sets requires nodes to exhibit regular connection patterns. For instance, if a set of k nodes are known to be always online from 8am to 8pm, and if a set of k other nodes are known to be online from 8pm to 8am, then a candidate set can be formed by combining these two sets of nodes. In the next section, we study two real-world traces and show that nodes actually exhibit regular connection patterns.

B. Connection regularity in real-world systems

Different works have studied node availability in distributed systems and have revealed that diurnal availability patterns are commonly exhibited [11] [13] [14] [15]. In the context of this work, we studied the connection regularity of nodes belonging to two different systems: eDonkey and Skype (details on the traces are provided in Section IV). We divide each days in 24 1-hour time slots and consider that a node is *online* in a given time slot t on day d if it connected at least once during the t^{th} hour of day d. Note that this coarse-grained granularity is required provided that some of the traces we are using are based on periodic pinging of nodes occurring every 30 minutes. We compute the regularity of node connections using the technique presented in [15]: a node is considered *regular* for a given time slot t if it connected in at least a percentage R of days of the trace at that time slot t. We tested two distinct values for R: 80% and 85% (R is called *regularity threshold*). We report the results we obtained in Figure 6 (eDonkey) and Figure 7 (Skype). Each graph contains 24 bars corresponding to the 24 1-hour time slots of the day. A bar of size X% for a time slot t means that, on average, X% of the nodes that are connected in time slot t are regularly connected in time slot t (i.e. they have a probability greater or equal to R to be connected in time slot t). Not surprisingly, we observe that increasing the regularity threshold R reduces the percentage of regular nodes. More interestingly, we observe that a quite large fraction of the nodes are regular. For instance, with a regularity threshold $R = 80\%$, we observe regularity values of 40% (eDonkey) and 25% (Skype).

C. Implementation

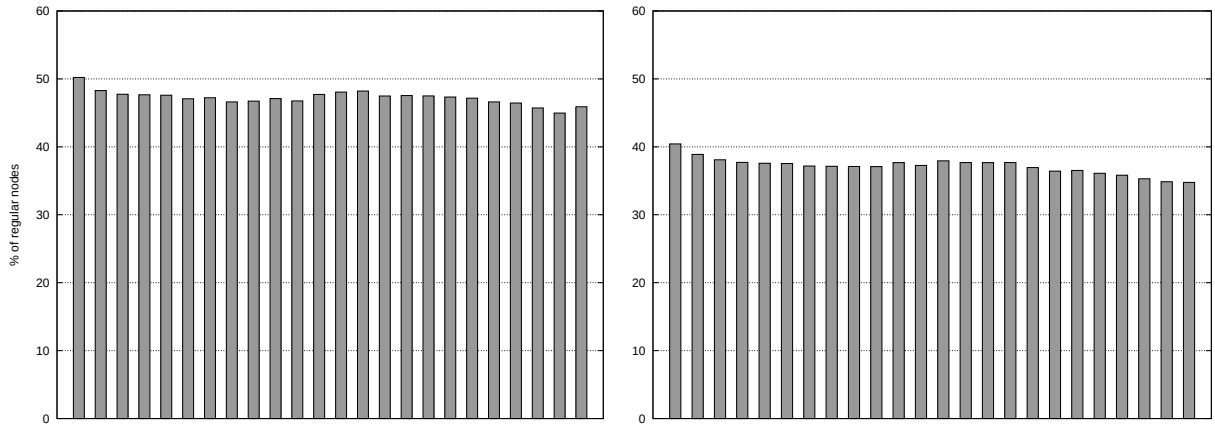
In this section, we describe our implementation of the *Regular* replication strategy. Each node maintains a log of its connections that it uses to compute its regularity, i.e. the probability that it be connected at each time slot of the day. This regularity data is exchanged among nodes using a gossip-based protocol à la Cyclon [18]: each node locally maintains a continuously changing partial view of nodes belonging to the system with which it exchanges informations about the connection regularity of nodes (including nodes that are *not* currently online). To ensure that exchanged informations are up-to-date, nodes associate a timestamp to the regularity information they disseminate. Each node stores the regularity informations it collects in a *regularity table*. This table contains 24 buckets, one per time slot. Each bucket contains a set of nodes that are regular in the associated time slot.

When a node creates an object to be stored in the DHT, it first creates a candidate set for that object using its regularity table. This simply consists in selecting as many nodes as required from each bucket, in order to ensure that there will be a high probability that the candidate set contain k online nodes at any time of the day. The node then creates a replica set for the new object that it fills with k currently online nodes taken from the candidate set. Both the candidate and the replica sets are replicated on the k first nodes in the DHT that have a higher identifier than the hash of the created object (this is similar to the replication of localization meta-data in the *Sticky* strategy presented in Section II).

The creating node then notifies the currently online nodes belonging to the candidate (resp. replica) set that they have been selected to be in the candidate (resp. replica) set of the created object. Nodes belonging to the candidate set periodically ping each others. Whenever they detect that the replica set contains fewer than k nodes, one currently online node belonging to the candidate set is randomly chosen to join the replica set. If the candidate set does not contain online nodes not currently in the replica set, a temporary node is chosen to be part of the replica set. This temporary node is chosen among the k nodes currently replicating the candidate and replica sets.

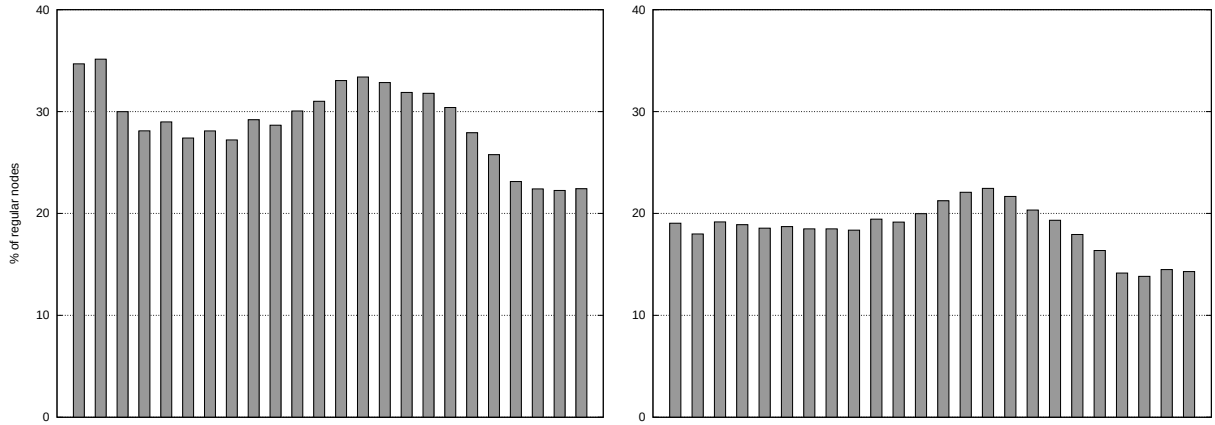
IV. Evaluation

In this section, we compare the performance achieved by the regular replication strategy to that achieved by



(a) $R = 80\%$ (b) $R = 85\%$

Fig. 6: **eDonkey**: percentage of regular nodes.



(a) $R = 80\%$ (b) $R = 85\%$

Fig. 7: **Skype**: percentage of regular nodes.

replication strategies described in Section II. We start by a description of the experimental settings, followed by a performance comparison using two traces from real-world systems: eDonkey and Skype. Finally, we present a scalability analysis of the *Regular* replication strategy using the eDonkey trace.

A. Experimental settings

We developed a C++ event-based simulator *a la* PeerSim [19]. For efficiency reasons, we do not simulate the routing protocol used in the DHT, which means that nodes do not maintain a routing table. We rather focus on simulating the replication protocol that is running among nodes.

Each node in the system stores 500MB of data that is split into 10MB chunks. We use the fact that data is split into chunks to allow migrating replicated objects from multiple sources in parallel. Moreover, in the experiments we run, we configured the replication factor to $k = 4$, which means that the replication protocol tries to keep 4 replicas of each chunk at any time. This replication factor was enough to ensure that no chunk was lost in all experiments we ran.

We take into account the fact that Internet nodes have a limited bandwidth by bounding the rate at which nodes can exchange data. More precisely, we consider that nodes have a maximum upload bandwidth of 1Mb/s. Moreover, in order to reduce the overall bandwidth usage, nodes keep the objects they replicate even after exiting the replica set. That way, a node that temporarily exits a replica set does not pay a high cost if it rejoins the replica set. Finally, to simulate realistic churn (i.e. node arrivals and departures), we use two traces from

real-world applications that are described below[1]:

- **eDonkey:** the trace [15] contains a 27-day log of user connections and disconnections to one of the main eDonkey servers. The traces gathers data for more than 14 millions users.
- **Skype:** the trace [21] contains a 1-month log of the connections and disconnections of 2000 Skype users. The trace was built by pinging nodes participating to the Skype superpeer network every 30 minutes for one month, beginning September 12th, 2005.

Two replication strategies rely on an analysis of past node connections (*Anti-correlated available* and *Regular*)[2]. Therefore, we split the simulations into two phases: the first phase lasts 20 days and is a learning period, after which the availability correlation and the connection regularity can be computed. The second phase lasts 7 days and is used to collect the presented results.

B. Performance comparison

The performance results obtained using the various replication strategies are reported in Table I (eDonkey) and Table II (Skype). In each table, we report the following metrics:

- *Total bandwidth (GB):* this metrics represents the total number of GB that are transferred by all nodes during the experiment (7 days). A good replication strategy is one that minimizes this value.
- *Max bandwidth per node (GB):* this metrics represents the maximum bandwidth (in GB) that is consumed by a node during the experiment (7 days). A good replication strategy is one that minimizes this value.
- *Max stored objects per node:* this metrics represents the maximum number of objects that is stored by a node during the experiment. We only count objects in the replica set of which the node is currently in. We do *not* count objects that are kept after the node exited the replica set. A good replication strategy is one that minimizes this value.
- *Perc. of nodes storing 50% of the overall data:* this metrics represents the percentage of the most loaded nodes in the network that store an aggregate value of half the data contained in the entire system. For instance, if this metrics equals 9%, this means that 9% of the nodes in the network (which are the most loaded ones) are responsible for the storage of half the data of the entire network. The larger this value, the better the replication strategy. Indeed, a large values means that the storage load is balanced on a large fraction of nodes.
- *Perc. of nodes that are replica at least once:* this metrics represents the percentage of nodes that, during the experience, act at least once as replica for an object. The larger this value, the better the replication strategy. Indeed, a large value indicates that the replication load is balanced on a large fraction of nodes.

Each line in Tables I and II corresponds to a replication strategy. For replication strategies relying on availability prediction, we use an oracle that outputs, at any time, how long every node will remain in the system. Moreover, considering the *Anti-correlated availability* strategy, we configure the number of pairs contained in each replica sets in such a way that there is always $k = 4$ replicas online[3]. Finally, regarding the *Regular* replication strategy, we use two different values for the regularity threshold: 80% and 85%.

The first observation we can make is that **the *Standard*, *Sticky* and *Anti-correlated availability* replication strategies induce a much higher total bandwidth consumption than other replication strategies, but better balance the replication and storage loads on nodes**. For instance, in the eDonkey trace, the *Standard* replication strategy uses 17.7 times more bandwidth than the *Regular* strategy with a regularity threshold of 85%. This higher bandwidth consumption translates into higher networking requirements for individual nodes: for instance, in the case of eDonkey, the maximum bandwidth that is used by a node is 7.7GB for the *Standard* strategy, whereas it is only 1.07GB for the *Regular* strategy with a regularity threshold of 85%. Regarding the replication load, we observe that with the *Regular* and *Sticky*, and *Anti-correlated availability* strategies, the number of nodes that are replica at least once is systematically higher than with other strategies (for all traces). Finally, regarding storage, we can observe that the *Standard*, *Sticky* and *Anti-correlated availability* strategies better balance storage than other strategies: for instance, in the eDonkey case, 36.5% of nodes are in

[1]These traces can be found on a public repository maintained by Godfrey [20].

[2]The *Most available* and *Sticky most-available* strategies should also rely on an analysis of past node connections. Nevertheless, we implement availability predictions using an oracle.

[3]Note that this configuration is different from that used in [12], where authors guarantee that there are k nodes in replica sets, but including temporarily offline ones.

	Total bandwidth (GB)	Max bandwidth per node (GB)	Max stored objects per node	Perc. of nodes storing 50% of the overall data	Perc. of nodes that are replica at least once
Standard	2812.2	7.7	40	36.5%	100.0%
Sticky	2436.9	10.0	40	33.3%	96.2%
Anti-correlated availability	2104.2	4.8	61	38.9%	97.7%
Most-available	574.0	11.9	77	7.5%	29.0%
Sticky most-available	290.6	11.7	77	6.7%	24.4%
Regular (R = 80%)	372.6	1.5	43	19.1%	75.6%
Regular (R = 85%)	159.1	1.1	57	15.5%	60.2%

TABLE I: **eDonkey:** evaluation of the different replication strategies (5000 nodes).

	Total bandwidth (GB)	Max bandwidth per node (GB)	Max stored objects per node	Perc. of nodes storing 50% of the overall data	Perc. of nodes that are replica at least once
Standard	2395.1	28.7	127	35.0%	100.0%
Sticky	2174.1	31.0	127	30.5%	96.2%
Anti-correlated availability	1884.9	23.1	121	29.8%	97.0%
Most-available	991.6	38.3	187	6.9%	35.0%
Sticky most-available	648.8	32.1	185	5.0%	25.3%
Regular (R = 80%)	790.8	18.0	136	8.8%	69.7%
Regular (R = 85%)	441.1	16.3	187	5.1%	51.2%

TABLE II: **Skype:** evaluation of the different replication strategies (2000 nodes).

charge of storing half of the data when the *Standard* replication strategy is used. This drops to 15.5% when the *Regular* strategy is used with a regularity threshold of 85%.

The second observation we can make is that **for the *Regular* strategy, increasing the regularity threshold *R* decreases the network usage, but negatively impacts the replication and storage load balancing**. For instance, we observe on the eDonkey trace that the *Regular* strategy induces 372.6GB of network traffic when $R = 80\%$, whereas it induces 159.1GB of traffic when $R = 85\%$. At the same time, with $R = 85\%$, 75.6% of the nodes are replica at least once, whereas only 60.2% of the nodes are replica at least once when $R = 85\%$. Similarly, with $R = 80\%$, 19.1% of the nodes are responsible for the storage of half the data, whereas it drops to 15.5% when $R = 85\%$. This observed behavior makes sense: increasing the regularity threshold R decreases the number of nodes that are in the candidate set. This does thus reduce the network load (fewer object migrations are necessary), but also negatively impacts the replication and storage load balancing.

The third observation we can make is that **the *Regular* replication strategy induces lower network usage, and better balances the replication and storage loads than the *Most-available* and *Sticky most-available* replication strategies**. Tables I and II show that, using the two traces considered in this performance study, it is possible to configure the regularity threshold so that the *Regular* strategy be less bandwidth consuming, and be better at balancing the replication and storage load than the *Most-available* and *Sticky most-available* strategies. For instance, in the eDonkey trace: *Regular* with $R = 85\%$ is better than *Most-available* and *Sticky most-available*. The better network usage is easily explained by the fact that, by design, the *Regular* strategy induces fewer object migrations. The better balancing of both the storage and replication loads is due to the fact that the set of nodes participating in replica sets with the *Sticky most-available* strategy is smaller than the set of nodes participating in replica sets with the *Regular* strategy. This phenomenon is clearly illustrated by values reported

in the last column of the Tables. Using *Regular* with $R = 85\%$, 60.1% of the nodes are replica at least once, whereas only 24.4% of the nodes are replica at least once when using the *Sticky most-available* strategy.

As a conclusion, we can say that if bandwidth consumption is not an issue, then the *Anti-correlated availability* strategy is the best choice. If bandwidth consumption is an issue (which, is considered the common case [10]), the *Regular* strategy is the best choice.

C. Scalability analysis

In this section, we assess the scalability of the *Regular* replication strategy. Using the eDonkey trace, we performed experiments involving 1000, 5000, 50.000, 100.000 and 250.000 nodes, respectively. In all cases, the regularity threshold R was set to 85%. We depict the evolution, as a function of the number of nodes, of the maximum bandwidth that is consumed per node (Figure 8) and of the maximum number of objects that a node stores (Figure 9). We see that these two metrics increase very slowly when the number of nodes increases. For instance, we observe that multiplying the size of the system by 250 less than double the maximum bandwidth consumed by a node and the maximum number of objects that are stored by a node.

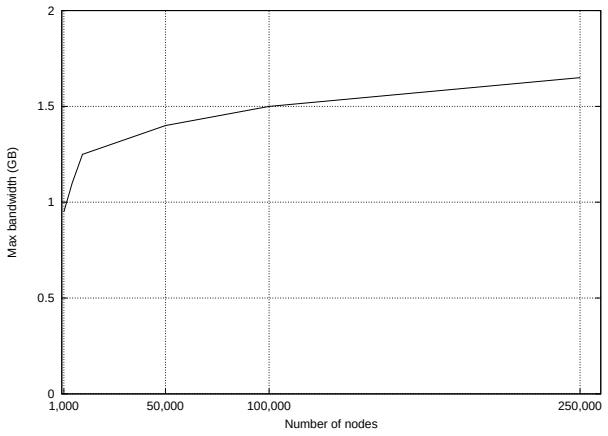


Fig. 8: **eDonkey:** maximum bandwidth consumed per node as a function of the number of nodes in the system.

V. Related work

Distributed Hash Tables (DHTs) (e.g. [1] [2]) have established in the last decade as a practical and effective approach to store and retrieve objects in a large and dynamic network of nodes. Their underlying principle relies on mapping (via variants of *consistent hashing* [7]) objects and nodes on a large numeric key-space, whose

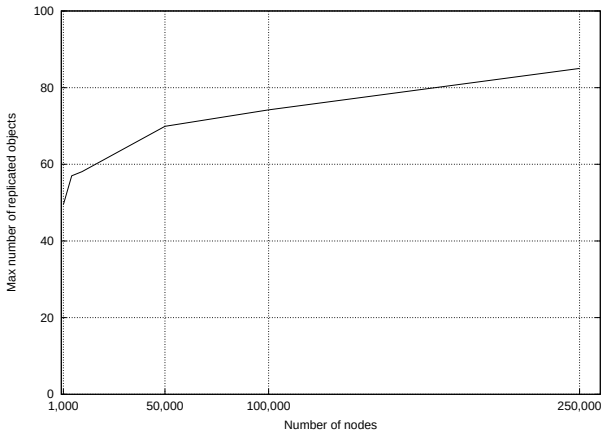


Fig. 9: **eDonkey:** maximum number of objects stored by a node as a function of the number of nodes in the system.

topology depends on the actual DHT implementation (e.g. a ring for Chord [1] and Pastry [2], a d-dimensional torus for CAN [22] and Tapestry [23]). The different topology variants share that fact that the node in charge of a stored object is the one whose position is the closest to that of the object on the key-space.

To ensure availability of stored objects in case of node failures, the first solutions introduced redundancy by replicating a stored object on the k closest (online) nodes on the DHT. This strategy is usually called *leafset-based replication* [8], and was first embodied by DHash [3] and Past [4]. This approach is captured by the *Standard* replication strategy which we described in Section II-A: a node in charge of an object replicates the latter on the first $k-1$ online successors on the ring.

Despite the fact that the *Standard* strategy ensures high availability of stored objects despite node arrivals and departures, it induces (possibly high) bandwidth consumption to migrate copies of the objects to the actual first k online successors on the ring. More flexible replication strategies have then be proposed. In RelaxDHT [17], an online node replicating an object is allowed to remain a replica as long as it falls into the first $S \geq k$ closest nodes. The location of the replicas is maintained in the form of object metadata, which are replicated and maintainted (via the *Standard strategy*) on the first k online nodes, allowing to lookup the actual object copies [16]. The *Sticky* replication strategy presented in [11], and which we have described in Section II-B, is a generalization of this approach: an online node is allowed to remain a replica despite its future higher distance on the ring.

In recenter years, more sophisticated strategies have

been proposed to further limit the number of objects migration by exploiting some knowledge about node availability. The first solutions assumed roughly homogeneous availability patterns among nodes (e.g. [24] [25]). Subsequent works have taken a finer control by focusing on individual nodes' availability [11] [15] [12]. In particular, in [11] the authors describe two replication strategies which rely on predictors in order to place an object copy on the most expected available nodes. These two replication strategies correspond in our study to the *Most-available* and *Sticky most-available* strategies, which we have detailed respectively in Section II-C and Section II-D. Finally, in a very recent work [12], the authors propose the anti-correlated replication strategy illustrated in Section II.

VI. Conclusion

Distributed Hash Tables (DHTs) provide a simple distribute put/get abstraction upon which it is possible to build efficient distributed storage systems. Object replication is typically used to ensure object availability in case of churn. Due to bandwidth costs incurred in object migration, bandwidth remains the most important and limited resource. In the last years, many works have focused on designing replication strategies which better utilize bandwidth. This has ranged from more flexible replica placement strategies, to adaptations accounting for average network and node metrics, and more recently to exploiting knowledge of individual node availability to guide replicas placement. We presented in this paper a replication strategy which, instead of leveraging the most available nodes, exploits the regularity connection pattern exhibited by nodes. We have evaluated the effectiveness of our solution by comparing it with five different existing replication strategies. Results show that the regularity-based replication strategy dramatically reduces bandwidth utilization and better balances the replication and storage loads among nodes.

Acknowledgements

We are grateful to Pierre-Louis Aublin, Sonia Ben Mokhtar, Francesco Bongiovanni, Fabien Gaud and Baptiste Lepers for their helpful feedback on this work.

References

[1] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, 2003.

[2] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of the International Conference on Distributed Systems and Platforms (Middleware)*, Heidelberg, Germany, November 2001, pp. 329–350.

[3] F. Dabek, J. Li, E. Sit, J. Robertson, F. Kaashoek, and R. Morris, "Designing a DHT for Low Latency and High Throughput," in *Proceedings of the International Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, USA, March 2004, pp. 85–98.

[4] A. Rowstron and P. Druschel, "Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility," in *Proceedings of the International Symposium on Operating System Principles (SOSP)*, Banff, Alberta, Canada, October 2001, pp. 188–201.

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, ser. SOSP '07, 2007, pp. 205–220.

[6] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 35–40, April 2010.

[7] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web," in *Proceedings of the International Symposium on Theory of Computing (STOC)*, El Paso, Texas, United States, May 1997, pp. 654–663.

[8] S. Ktari, M. Zoubert, A. Hecker, and H. Labiod, "Performance evaluation of replication strategies in dhts under churn," in *Proceedings of the International Conference on Mobile and Ubiquitous Multimedia (MUM)*, Oulu, Finland, December 2007, pp. 90–97.

[9] Q. Lian, W. Chen, and Z. Zhang, "On the Impact of Replica Placement to the Reliability of Distributed Brick Storage Systems," in *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, Columbus, OH, USA, June 2005, pp. 187–196.

[10] C. Blake and R. Rodrigues, "High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two," in *Proceedings of the International Workshop on Hot Topics in Operating Systems (HotOS)*, Lihue, Hawaii, USA, May 2003, pp. 1–6.

[11] J. W. Mickens and B. D. Noble, "Exploiting Availability Prediction in Distributed Systems," in *Proceedings of the International Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, USA, May 2006, pp. 73–86.

[12] A. Kermarrec, E. Le Merrer, G. Straub, and A. Van Kempen, "Availability-Based Methods for Distributed Storage Systems," 2010, INRIA Technical Report, http://hal.archives-ouvertes.fr/docs/00/57/41/38/PDF/Availability.pdf.

[13] R. Bhagwan, S. Savage, and G. M. Voelker, "Understanding Availability," in *IPTPS*, 2003, pp. 256–267.

[14] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs," in *SIGMETRICS*, 2000, pp. 34–43.

[15] S. Le-Blond, F. Le Fessant, and E. L. Merrer, "Finding Good Partners in Availability-Aware P2P Networks," in *Proceedings of the International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, Lyon, France, November 2009, pp. 472–484.

[16] K. Chen, L. Naamani, and K. Yehia, "miChord: decoupling object lookup from placement in DHT-Based Overlays," 2003, http://www.loai-naamani.com/Academics/miChord.htm.

[17] S. Legtchenko, S. Monnet, P. Sens, and G. Muller, "Churn-resilient replication strategy for peer-to-peer distributed hash-tables," in *In Proceedings of Stabilization, Safety, and Security of Distributed Systems, 11th International Symposium, SSS 2009, Lyon, France, November 3-6, 2009.*, ser. SSS '09, 2009, pp. 485–499.

[18] S. Voulgaris, D. Gavidia, and M. van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *J. Network Syst. Manage.*, vol. 13, no. 2, 2005.

[19] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris, "The Peersim simulator," http://peersim.sf.net.
[20] B. Godfrey, "Repository of availability traces," http://www.cs.uiuc.edu/homes/pbg/availability/.
[21] S. Guha, N. Daswani, and R. Jain, "An Experimental Study of the Skype Peer-to-Peer VoIP System," in *Proceedings of the International Workshop on Peer-to-Peer Systems (IPTPS)*, Santa Barbara, CA, USA, February 2006.
[22] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proceedings of the International SIGCOMM Conference*, San Diego, CA, USA, August 2001, pp. 161–172.
[23] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: a Resilient Global-Scale Overlay for Service Deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, 2004.
[24] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total Recall: System Support for Automated Availability Management," in *Proceedings of the International Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, USA, March 2004, pp. 337–350.
[25] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris, "Efficient Replica Maintenance for Distributed Storage Systems," in *Proceedings of the International Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, USA, May 2006.